

An Approach to the Design of a Page Description Language

DAVID J. HARRIS
Chelgraph Limited

ABSTRACT

With the recent development of cheap highly functional laser printers and Raster Image Processors, there has been an upsurge of interest in languages for interfacing to these devices. An approach to the design of such a Page Description Language is described, the primary design requirement being a clean interface which is an easy target for translators from various front-end systems. The design of an actual PDL, the Chelgraph ACE language, based on these principles is described. Finally the ACE language is reviewed in the light of experience gained in its use.

1. Introduction

This paper discusses some issues relevant to the design of a Page Description Language (PDL). A PDL is a type of language commonly used for communicating page information from a composition system to an intelligent page printer. These languages are usually specified by the printer manufacturer as an input language, but device-independent outputs from some composition packages, for example DI-TROFF, are also PDLs. I also present a particular PDL, the ACE language, which has been designed by us at Chelgraph and implemented on our Raster Image Processor, the features of ACE itself have been described elsewhere [Chel84, Harris84].

2. What Is a PDL ?

There have always been languages for communicating page information to typesetters and printers. Until recently the capabilities of computer output printers have been very limited, so their input languages have been simple ASCII formats modified by escape codes. Typesetters such as the Autologic APS 5, on the other hand, have quite complex input languages with a syntax and tens of commands.

With the recent advent of laser printers and raster typesetters the issue of PDLs has received much attention, and a number of languages have been invented and often promulgated as industry standards. Laser printers and

modern typesetters usually consist of two parts, a raster marking engine whose task it is to make marks on the paper, and a Raster Image Processor (RIP) which takes in some description of a page (i.e. a PDL) and generates a bit-image of the page for the marking engine. The facilities incorporated in the typical modern RIP, which can usually scale and rotate characters and generate other graphics, together with the need to interface these printers to a wide variety of composition packages and other 'front ends', make the specification of a good PDL a real challenge.

3. Design Aims

The principal reason for having a PDL is to provide a clean interface to a typesetter or other printing machine, other secondary aims can include the provision of device-independence and the use of the PDL as a 'metafile' to transfer page descriptions between sites. Although the secondary, and grander, aims are sometimes seen as important, the provision of a good composer/printer interface must remain paramount.

It is desirable that the PDL should be easy to implement, mainly for those whose task it is to generate it from a front end system, but it must also be easy to implement at the printer end if it is to become widely accepted (i.e. an 'industry standard'). In order to provide this ease of interfacing the language must be designed with the needs of the front-end systems in mind, drawing on existing standards, both formal and informal, used in the field.

A further requirement, from the point of view of the printer supplier, is that the PDL provides full access to the facilities of the printer and presents a logical, and easily understandable model of the printer to the user of the language.

Additional aims, good for the design of any language, include Consistency, Orthogonality, Simplicity and Efficiency (these are of course, usually incompatible). Readability of the language by humans is not a prime concern as PDLs are normally generated and interpreted by programs, readability does greatly help the writers of such programs, however.

4. The Three Layers of Language

Document content can be defined at several levels, these seem to break most naturally into three layers, each layer having its own characteristic ways of defining a document.

On the highest layer, we have Abstract languages. These are often declarative in tone (for example 'this is a paragraph'), and are very portable, both between output devices and even different composition systems. A good example of this type of language is the SGML (Standard Generalised Markup Language). Another, more accessible, example is the TBL table description

language used with the TROFF documentation programs.

On the medium layer are the Procedural languages, which tend to have a command and parameter structure (for example 'at depth X change margin to Y'). They are seldom portable, but can offer output-device independence. An example from this layer is CORA V, the Linotype composition language.

The lowest layer contains the Slave languages. On this layer all abstract information is gone (for example 'goto X,Y'). All decisions, such as kerning, line breaks, and footnote placement have been made, although general coordinate transformations are still possible (for example rotation and scaling of the whole page). An example of a Slave language is the TeX DVI output format.

Document composition is often seen as being analogous to compilation [Reid80]. Using this model the higher two layers parallel various types of high level languages, and the Slave level the compiler intermediate code or the machine code.

The most appropriate layer for PDLs is the Slave layer. The reason for this is to provide the maximum ease of interfacing. It is impossible to translate from a low layer to a higher layer, and usually difficult to translate between languages on the same layer. On the other hand it is straightforward to translate from a high layer to a low layer, a low layer language thus provides the easiest 'target' for the output routines from composition programs.

Using the slave layer can, although aiding one of the primary design aims (ease of interface), work against one of the secondary aims (portability). When documents must be ported between printers and systems, formatting decisions may need to be re-interpreted in the light of the facilities of the target printer (for example available fonts). Such portability can best be provided by a language on the Abstract layer, where style tags can be redefined at will. The question of portability must not be allowed to confuse the design of a Slave PDL, as it can be dealt with far better in the higher layers.

5. The Design of ACE

The reason for developing ACE was to provide an interface language for the Chelgraph Modular RIP. This is a flexible Raster Image Processor whose modular design allows it to be easily configured to drive almost any raster device, from an A4 300 dot per inch laser printer to a 2000+ dot per inch double broadsheet laser platemaker. This flexibility, coupled with the RIPs advanced functions combining versatile (typesetter-quality) text and computer graphics meant that the RIP must be interfaced to a wide variety of driving front end systems, ranging from newspaper editorial systems, to PC based graphics packages.

The starting point for the design, under the guidance rules developed

above, was to consider existing standards formal and de facto. In the case of computer-generated graphics there was in existence a very suitable formalised standard in the proposed ANSI Virtual Device Metafile (VDM). This standard (which has now been taken up by ISO as CGM, Computer Graphics Metafile) is part of the GKS family of standards for graphics. It specifies content at a level equivalent with the Slave PDL, thus making it suitable. CGM is an evolving standard which specifies line and area drawing functions such as :

- Polyline
- Arc
- Elliptical Arc
- Polygon
- Circle
- Circular Arc Close
- Elliptical Arc Close

All these commands can be modified by a set of modal variables to produce different effects (such as tinted or cross-hatched circles). CGM also provides a model for specifying text attributes and the interaction of text with graphical objects. For example text rotation angle is specified by the text vector, which is a vector running along the text baseline in the direction of travel of the text. In the corresponding ACE command 'tv 0 1' (being a horizontal vector) is the default, this means that the baseline of text runs horizontally from left to right. By specifying 'tv 1 1' the text would be rotated so that the baseline runs at 45 degrees to the horizontal. Rotation can be combined with shearing, reflection and scaling to create a wide range of effects.

When it comes to the lay-down of text however, CGM with its background in graphics systems labelling pie diagrams and graphs is not really suitable for bulk text formatted by a typesetter. There are also no helpful formal or industry standards in the typesetting field. In the specification of this area of ACE we drew on experience gained in typesetter interfacing. This suggested that the important point in order to provide a clean, trouble-free interface was that ACE should itself specify the position of each character on the page, thus the reader of the ACE does not need access to width tables in order to interpret it. This kind of output can always be generated easily from a composition system's model of the page and helps to reduce the number of copies of a width table that need to exist within a system (hopefully to one). We were further influenced in this direction by the DI-TROFF intermediate device independent code specification [Kern82].

As in the DI-TROFF code system, ACE uses a scheme whereby escapements (or relative motions) are included in the file along with the text characters. Absolute positioning commands are also available, but are seldom used. So, to set the line 'The ACE language' the code could be :

```
M "T28h23e38A28C28E47l11a21n24g19u24a21g19e. G
V 48
```

The 'M' and 'G' commands save and restore the current position and the 'V' command applies a relative motion at 90 degrees from the text vector (i.e. in the correct direction for inter-line spacing). Each character of text is followed by, in this case, two digits of escapement information. ACE provides a means whereby this may be reduced to one in most cases, by using offsets and number bases greater than 10. The units of measurement need not be device pixels.

A further boon of this way of positioning characters is that the composition system does all the justification, kerning and letterspacing, and other aesthetic positioning calculations. There are an innumerable number of ways of doing these and to provide for them efficiently in the Slave PDL requires a number of fairly complex command facilities. These things really are done much better in the front-end system or its output driver. Of course, a charge of inefficiency can be made against this approach but it is greatly mitigated the relative positioning and escapement encoding schemes. Any remaining inefficiency is a small price to pay for simplicity.

6. Implemented ACE processors

As mentioned above the first receiver of ACE was the Chelgraph RIP, this has become a successful product and is being designed in to laser printing and typesetting equipment.

On the side of generating ACE the simplicity of its approach has proved itself again and again. Chelgraph and its associates, with only a small development team, have already implemented drivers for ACE from typesetting languages : Linotype's CORA V, Autologic's APS 5 ICL and Monotype's Lasercomp; microcomputer packages : Digital Research GSX and GEM, and Microsoft Word; and DI-TROFF (including PIC, EQN and TBL).

The adherence of ACE to graphics standards often helps (e.g in the case of GSX, which is also derived from VDM), and the similarity with DI-TROFF made that driver easy. Where neither of these holds ACE generation is still straightforward due to its simple model of character positioning and page construction.

7. Conclusions

Our experience has shown that, in general, the right decisions were made in the design as borne out by the ease of development of the ACE drivers. As further testimony to the design, when the APS 5 emulator was developed it was desired to provide a 'wrong reading' (reflected output) facility in the emulator. This was implemented simply by the emulator modifying the setting of the ACE text vector and up vector values, no new commands had to be added to ACE.

To draw some conclusions on the form of the resulting language I will compare it with PostScript [Adobe84], which is certainly the most discussed PDL at the moment. A comparison of output functions available in the two languages (ACE and PostScript) certainly shows that there are some functions in PostScript, for example the ability to image bitmap graphics, which are not present in ACE. Such a comparison does not, however, reach the heart of the difference between the two languages, as it would be easy to add commands to ACE to make such output functions possible.

The real difference lies in the number of 'high level' functions included in the language. The aim of ACE is to provide a simple, but sufficient, set of tools to make page images, rather like a machine instruction set. PostScript on the other hand provides programming language features such as functions, loops and variables. At first sight the PostScript approach may seem obviously the more desirable - Have not programmers progressed over the years from machine code, through assembler to 'C', LISP and Ada ? To use this analogy, attractive though it may seem, is to forget that the 'programmer' that generates the PDL is almost invariably a computer program and not a person.

Thus we come back to the analogy of PDL as a machine instruction set with document formatter or other composition software as the compiler. Those who design complex PDLs are effectively following the same road as mainframe and mini computer designers who built machines with large and complex instruction sets containing 'high-level' instructions. In the case of machine architectures it turned out that many of these instructions were little used by the compilers, a discovery that led to today's movement towards simpler RISC instruction sets. As is the case with instruction sets, a larger, more complex PDL tends to require more computer hardware (memory, processing power) to implement, at the same performance level and moment in time, than a simpler language, thus one should not ignore the benefits of simplicity.

To conclude, then, ACE provides (like the RISC machine) a simple set of basic operations which allow the higher layers of document preparation software to access device facilities through a consistent interface. These

higher layers incorporate programming features and add the bulk of the device independence into the document structure. To build such features in at the PDL level, I believe is unnecessary and leads to an uncalled for increase in the size and complexity of the language.

Acknowledgements

I would like to thank the referees, and in particular Heather Brown, for their comments on this paper.

References

- [1] Adobe Systems (1984). *PostScript Language Manual* (first edition), Adobe Systems, Palo Alto, California, 1984. PostScript is a trademark of Adobe Systems.
- [2] Chelgraph Ltd. *ACE (ASCII Coded Escapement Language) Specification*, obtainable from Chelgraph in Cheltenham by post at the price of ten pounds sterling (plus VAT for UK orders).
- [3] Harris, D. J. (1984). *ACE - A Device Independent Lay Down Language for Text and Graphics*. In *PROTEXT I*, ed. J.J.H. Miller. Dublin, Ireland: Boole Press.
- [4] Kernighan B. W. (1982) *A Typesetter-independent TROFF*, Bell Laboratories, Murray Hill, New Jersey.
- [5] Reid B. K. (1980). *Scribe: a document specification language and its compiler*, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University.